

Rubens Rodrigues Teixeira Filho 11021EMT012

## **Exemplos Sistemas Eletropneumáticos Controlados por Arduino**

Uberlândia

25 de dezembro de 2019

Rubens Rodrigues Teixeira Filho 11021EMT012

## **Exemplos Sistemas Eletropneumáticos Controlados por Arduino**

Universidade Federal de Uberlândia - UFU

Faculdade de Engenharia Mecânica

Graduação em Engenharia Mecatrônica

Professor: José Jean

Uberlândia

25 de dezembro de 2019

# LISTA DE ILUSTRAÇÕES

Figura 1.1 – Circuito Eletropneumático . . . . .	6
Figura 1.2 – Circuito eletropneumático real. . . . .	6
Figura 2.1 – Esquema da montagem da Comunicação Serial . . . . .	7
Figura 2.2 – Montagem da Comunicação Serial . . . . .	7
Figura 3.1 – Esquema da montagem da Comunicação I2C . . . . .	12
Figura 3.2 – Montagem da Comunicação I2C . . . . .	13
Figura 4.1 – Esquema da montagem da Comunicação Ethernet . . . . .	18
Figura 4.2 – Montagem da Comunicação Ethernet . . . . .	19
Figura 5.1 – Esquema da montagem da Comunicação Wireless . . . . .	25

# LISTA DE CÓDIGOS

Código 2.1 – Corpo principal - Comunicação Serial . . . . .	8
Código 2.2 – Funções do controle da válvula - Comunicação Serial . . . . .	10
Código 2.3 – Informação dos Sensores - Comunicação Serial . . . . .	10
Código 2.4 – Acionamento via Serial - Comunicação Serial . . . . .	11
Código 3.1 – Corpo principal Mestre - I2C . . . . .	13
Código 3.2 – Acionamento via Serial Mestre - I2C . . . . .	15
Código 3.3 – Controle Válvula Mestre - I2C . . . . .	16
Código 3.4 – Informação Sensores Mestre - I2C . . . . .	16
Código 3.5 – Corpo Principal Escravo- I2C . . . . .	17
Código 4.1 – Corpo principal Emissor - Ethernet . . . . .	19
Código 4.2 – Envio Mensagem Emissor - Ethernet . . . . .	21
Código 4.3 – Corpo principal Receptor - Ethernet . . . . .	22
Código 4.4 – Controle Válvula Receptor - Ethernet . . . . .	23
Código 4.5 – Acionamento via Serial Receptor - Ethernet . . . . .	24
Código 5.1 – Corpo principal Emissor - Wireless . . . . .	26
Código 5.2 – Envio Mensagem Emissor - Wireless . . . . .	27
Código 5.3 – Corpo principal Receptor - Wireless . . . . .	28

# SUMÁRIO

<b>1</b>	<b>FUNCIONAMENTO DO SISTEMA</b>	<b>5</b>
<b>2</b>	<b>COMUNICAÇÃO SERIAL</b>	<b>7</b>
<b>3</b>	<b>COMUNICAÇÃO I2C DO TIPO MESTRE-ESCRAVO</b>	<b>12</b>
<b>3.1</b>	<b>Mestre</b>	<b>13</b>
<b>3.2</b>	<b>Escravo</b>	<b>17</b>
<b>4</b>	<b>COMUNICAÇÃO ETHERNET</b>	<b>18</b>
<b>4.1</b>	<b>Emissor</b>	<b>19</b>
<b>4.2</b>	<b>Receptor</b>	<b>22</b>
<b>5</b>	<b>COMUNICAÇÃO WIRELESS</b>	<b>25</b>
<b>5.1</b>	<b>Emissor</b>	<b>26</b>
<b>5.2</b>	<b>Receptor</b>	<b>28</b>
	<b>REFERÊNCIAS</b>	<b>29</b>

# 1 FUNCIONAMENTO DO SISTEMA

Nessa apostila será demonstrada diversas formas de comunicação entre Arduinos para o controle de um sistema eletropneumático. Em todos os exemplos que serão descritos, o sistema sempre deverá respeitar o mesmo comportamento, independente da forma de comunicação, e é descrito a seguir:

- O sistema deve iniciar avançado;
- O sistema deve recuar caso o botão seja pressionado ou o comando '1' seja enviado via serial(quando possível);
- Após o recuo do sistema ele deve aguardar 4 segundos e avançar automaticamente;
- Caso uma das condições de recuo seja acionada enquanto o sistema estiver recuado ele deve reiniciar a contagem do tempo;
- Caso o comando '0' seja enviado via serial o sistema deve avançar imediatamente;
- O sistema deve mostrar as informações dos sensores a cada 3 segundos (quando possível);

O circuito eletropneumático responsável por esse funcionamento pode ser visto na [Figura 1.1](#), e será o mesmo para todos os exemplos. As diferenças existentes ocorrerão nas montagens entre a comunicação entre os Arduinos e serão exibidas ao longo dos exemplos.

Todos os exemplos foram desenvolvidos com a configuração da entrada do tipo INPUT\_PULLUP, que define a entrada como padrão em nível logico alto, então para a sua ativação é necessário conecta-la ao terra da placa. Os códigos dos exemplos que serão carregadas nos Arduinos, em sua maioria foram divididos em diferentes arquivos para criar uma maior abstração do seu funcionamento, tornando a compreensão e a leitura do corpo principal mais agradável e de maior compreensão, além de tornar possível o reaproveitamento de códigos semelhantes.

Para dividir o seu código em diferentes arquivos o código principal deve possuir o mesmo nome do diretório onde ele esta. Na IDE do Arduino ao salvar um código ele automaticamente cria um diretório de mesmo nome e o coloca nesse diretório. Basta adicionar os arquivos de códigos a esse diretório que durante o carregamento para a placa eles também serão compilados. Para criar um arquivo dentro do mesmo diretório na IDE do Arduino basta utilizar o comando Ctrl+Shift+N

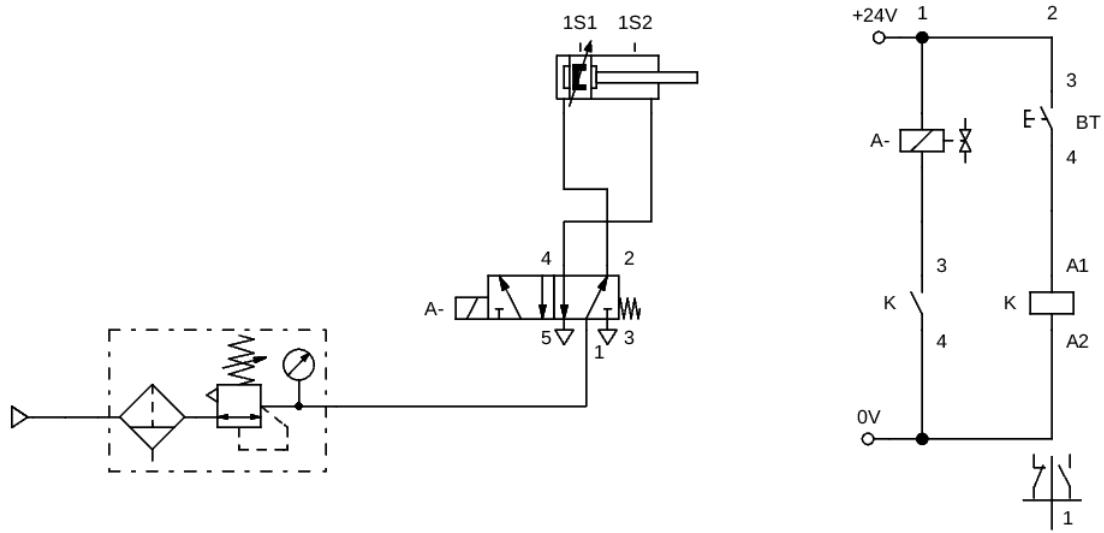


Figura 1.1 – Circuito Eletropneumático



Figura 1.2 – Circuito eletropneumático real.

Como pode ser visto na [Figura 1.1](#) o relé é responsável pela ativação do solenoide da válvula, que por sua vez aciona o recuo do cilindro, e por se tratar de uma válvula com retorno por mola, o sinal deve ser mantido pelo período necessário para que o sistema se mantenha recuado. Em nossos exemplos a saída do nosso sistema será um sinal de ativação no relé.

Em todos os exemplos mostrados as entradas foram conectadas em normal aberto, e foram conectadas ao mesmo terra. Na [Figura 1.2](#) pode ser vista a montagem real do circuito eletropneumático.

## 2 COMUNICAÇÃO SERIAL

Neste exemplo será demonstrada uma forma de realizar a comunicação serial entre o PC e o Arduino. Um exemplo da montagem física pode ser visto na [Figura 2.1](#), e a montagem real pode ser vista na [Figura 2.2](#).

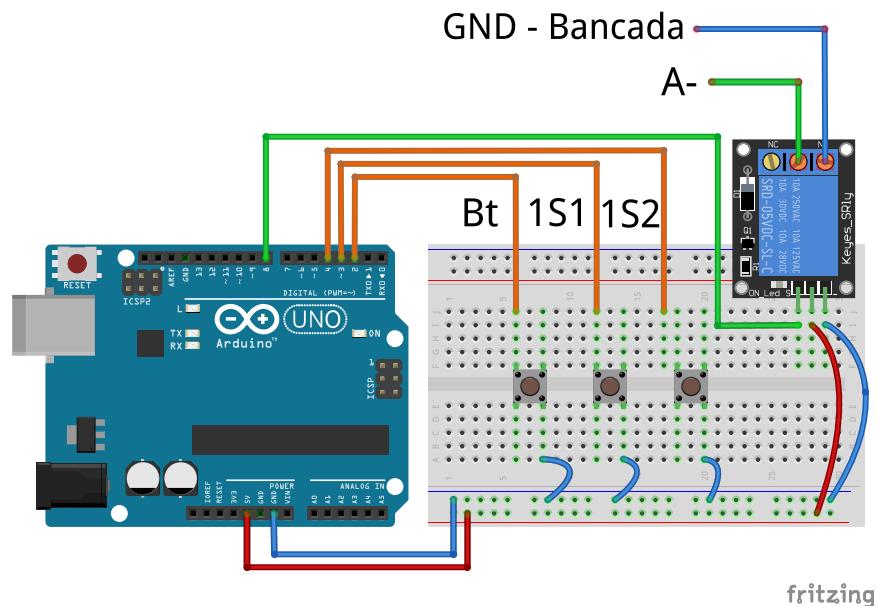


Figura 2.1 – Esquema da montagem da Comunicação Serial

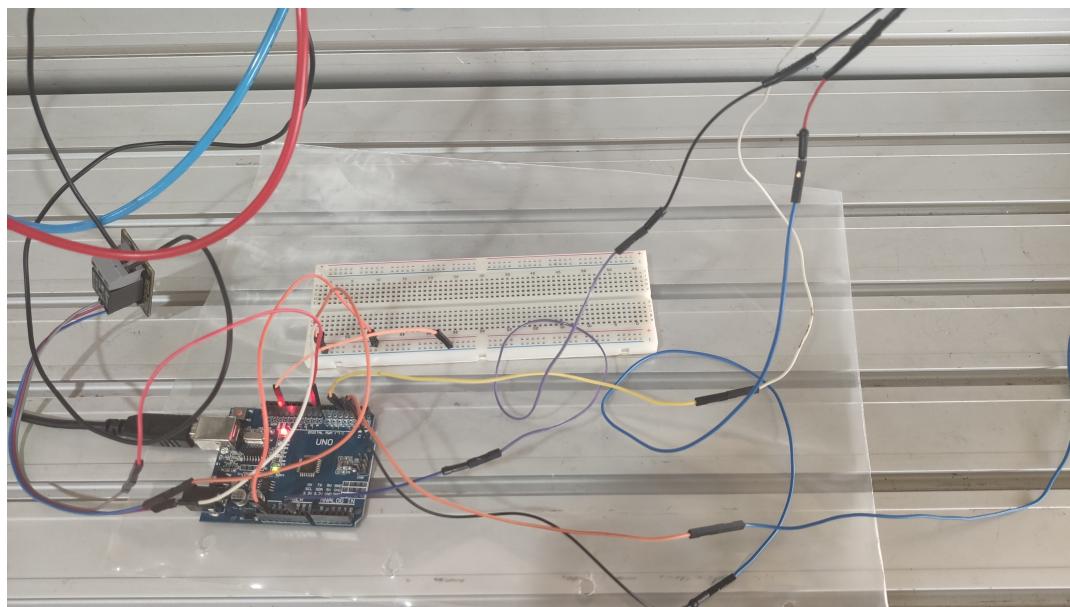


Figura 2.2 – Montagem da Comunicação Serial

Os códigos foram divididos em diferentes arquivos e são exibidos a seguir.

Código 2.1 – Corpo principal - Comunicação Serial

```
1 #include <TimerOne.h>
2
3 const int bt = 2;
4 const int a1 = 4;
5 const int a0 = 3;
6 const int rele = 8;
7 volatile bool flag = false;
8 volatile unsigned long tempo_disparo = 0;
9 int ultestbt;
10
11 void setup() {
12     // put your setup code here, to run once:
13     pinMode(bt, INPUT_PULLUP);
14     pinMode(a0, INPUT_PULLUP);
15     pinMode(a1, INPUT_PULLUP);
16     pinMode(rele, OUTPUT);
17     Serial.begin(9600);
18
19 Timer1.initialize(3000000);
20 Timer1.attachInterrupt(status_sensores);
21 }
22
23 void loop() {
24     // put your main code here, to run repeatedly:
25     int btt= digitalRead(bt);
26     int a00= digitalRead(a0);
27     int a11= digitalRead(a1);
28
29     if (btt != ultestbt) {
30         ultestbt=btt;
31         if(btt==0) {
32             recuo();
33         }
34     }
35
36     if(a00==0) {
37         if(flag==false) {
```

```
38         tempo_disp();  
39     }  
40 }  
41  
42 if (flag) {  
43     if (millis() - tempo_disparo >= 4000) {  
44         avanco();  
45     }  
46 }  
47 }
```

## Código 2.2 – Funções do controle da válvula - Comunicação Serial

```
1 void recuo() {
2     digitalWrite(rele, HIGH);
3     if(flag==true) {
4         tempo_disparo = millis();
5     }
6 }
7
8 void avanco() {
9     digitalWrite(rele, LOW);
10    flag=false;
11 }
12
13 void tempo_disp () {
14    flag=true;
15    tempo_disparo = millis();
16 }
```

## Código 2.3 – Informação dos Sensores - Comunicação Serial

```
1 void status_sensores() {
2     Serial.print("Sensor_A0:_");
3     Serial.print(digitalRead(a0));
4     Serial.print("\t");
5     Serial.print("Sensor_A1:_");
6     Serial.println(digitalRead(a1));
7 }
```

## Código 2.4 – Acionamento via Serial - Comunicação Serial

```
1 void serialEvent() {  
2     while (Serial.available()) {  
3         char ls = (char)Serial.read();  
4         Serial.print("Recebido:_");  
5         Serial.print(ls);  
6         Serial.print("\t");  
7  
8         switch (ls) {  
9             case '1':  
10                 recuo();  
11                 Serial.println("Comando_recuo_cilindro");  
12                 break;  
13             case '0':  
14                 avanco();  
15                 Serial.println("Comando_avanco_cilindro");  
16                 break;  
17             default:  
18                 Serial.println("Comando_invalido");  
19                 break;  
20         }  
21     }  
22 }
```

### 3 COMUNICAÇÃO I2C DO TIPO MESTRE-ESCRAVO

Nessa configuração um dos Arduinos será responsável pela coleta dos inputs, e o outro pelo acionamento do relé, o esquema físico da montagem é exibido na [Figura 3.1](#) e a montagem real pode ser visto na [Figura 3.2](#).

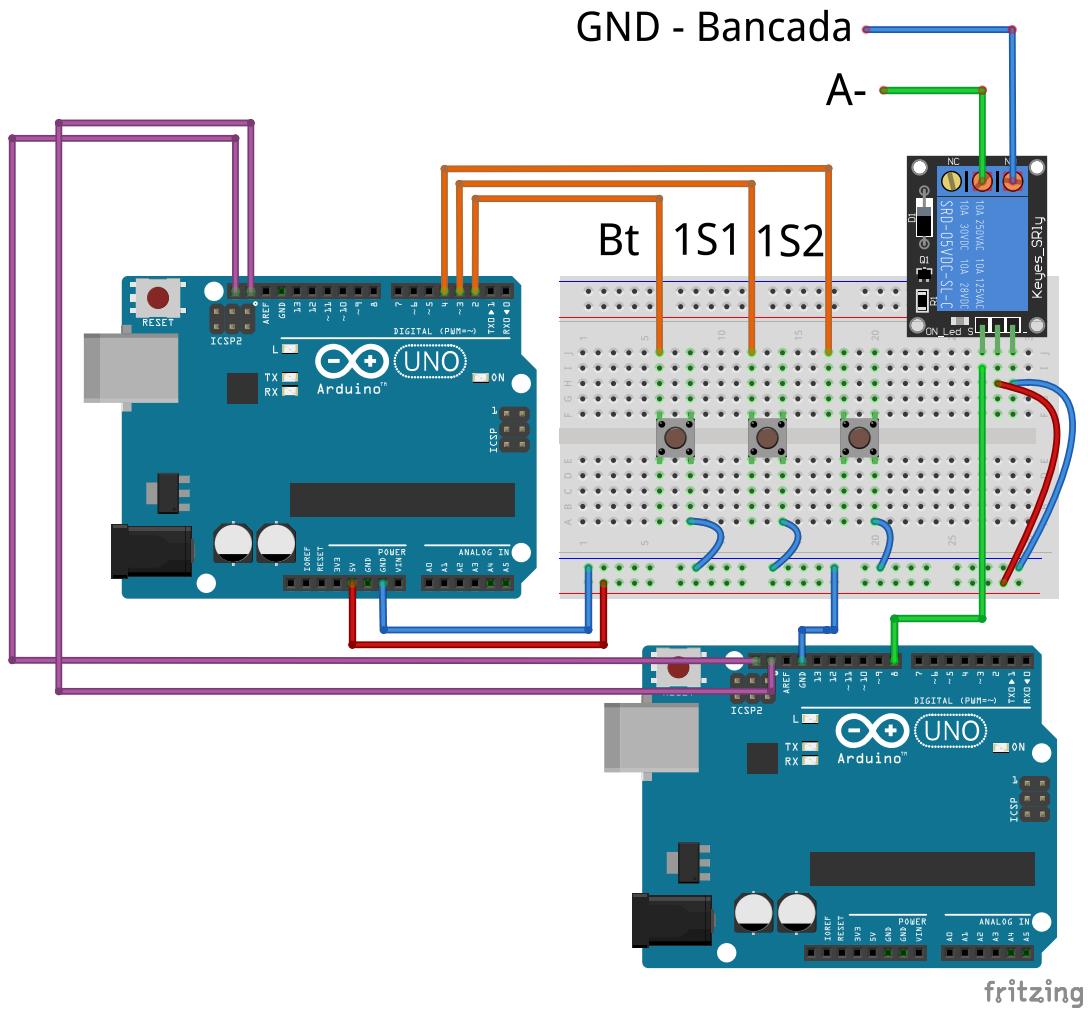


Figura 3.1 – Esquema da montagem da Comunicação I2C

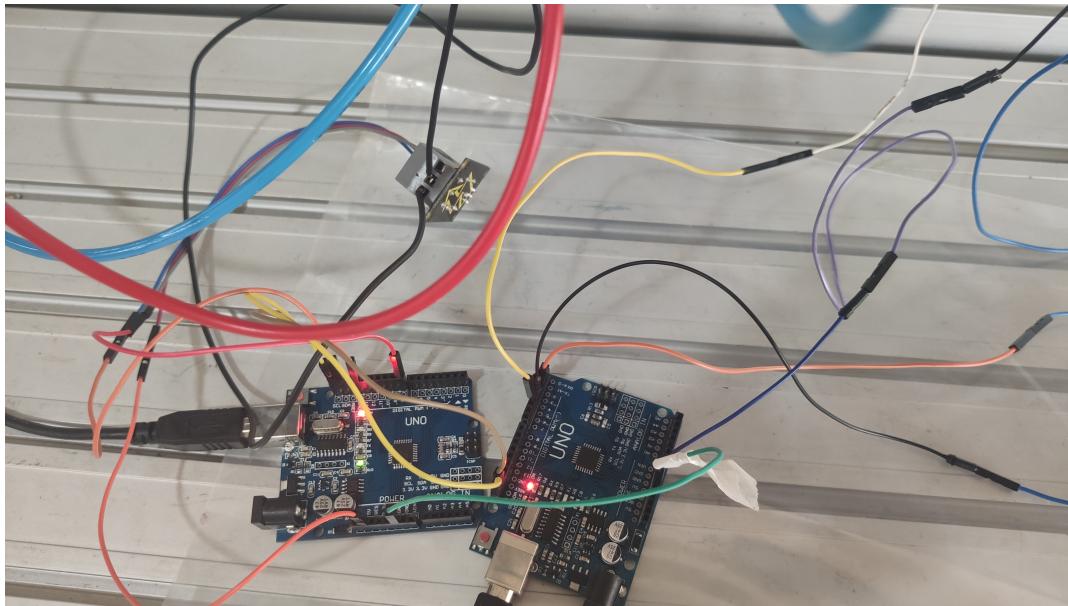


Figura 3.2 – Montagem da Comunicação I2C

### 3.1 Mestre

O mestre ficará responsável pelo acionamento da válvula, e os códigos utilizados são exibidos a seguir:

Código 3.1 – Corpo principal Mestre - I2C

```
1 #include <Wire.h>
2 #include <TimerOne.h>
3
4 const int rele = 8;
5
6 int bt,a0,a1,ultestbt;
7 unsigned long tempo_disparo = 0;
8 bool flag = false;
9
10
11 void setup() {
12     // put your setup code here, to run once:
13     Wire.begin();          //Inicia a biblioteca wire e define o
                           //dispositivo como mestre
14     Serial.begin(9600);    //Iniciando a comunicacao Serial
15     pinMode(rele, OUTPUT); // define o led como saida
16
17     Timer1.initialize(3000000);
18     Timer1.attachInterrupt(status_sensores);
```

```
19 }
20
21 void loop() {
22     // put your main code here, to run repeatedly:
23     Wire.requestFrom(2,3);      //Requisita 4 bytes do escravo 2
24     while (Wire.available()) {  //Leitura dos dados enviados pelo
                                  escravo
25         bt = Wire.read();
26         a0 = Wire.read();
27         a1 = Wire.read();
28     }
29
30     if (bt != ultestbt) {
31         ultestbt=bt;
32         if(bt==0) {
33             recuo();
34         }
35     }
36
37
38     if(a0==0) {
39         if(flag==false) {
40             tempo_disp();
41         }
42     }
43
44     if (flag) {
45         if(millis()-tempo_disparo >= 4000) {
46             avanco();
47         }
48     }
49 }
```

## Código 3.2 – Acionamento via Serial Mestre - I2C

```
1 void serialEvent() {  
2     while (Serial.available()) {  
3         char ls = (char)Serial.read();  
4         Serial.print("Recebido:_");  
5         Serial.print(ls);  
6         Serial.print("\t");  
7  
8         switch (ls) {  
9             case '1':  
10                 recuo();  
11                 Serial.println("Comando_recuo_cilindro");  
12                 break;  
13             case '0':  
14                 avanco();  
15                 Serial.println("Comando_avanco_cilindro");  
16                 break;  
17             default:  
18                 Serial.println("Comando_invalido");  
19                 break;  
20         }  
21     }  
22 }
```

## Código 3.3 – Controle Válvula Mestre - I2C

```
1 void recuo() {
2     digitalWrite(rele, HIGH);
3     tempo_disparo = millis();
4 }
5
6 void avanco() {
7     digitalWrite(rele, LOW);
8     flag=false;
9 }
10
11 void tempo_disp () {
12     flag=true;
13     tempo_disparo = millis();
14 }
```

## Código 3.4 – Informação Sensores Mestre - I2C

```
1 void status_sensores() {
2     Serial.print("Sensor_A0:_");
3     Serial.print(a0);
4     Serial.print("\t");
5     Serial.print("Sensor_A1:_");
6     Serial.println(a1);
7 }
```

## 3.2 Escravo

O Escravo será responsável pelas entradas do sistema.

Código 3.5 – Corpo Principal Escravo- I2C

```
1 #include<Wire.h>
2
3 const int bt = 2;
4 const int a1 = 4;
5 const int a0 = 3;
6
7 void setup() {
8     // put your setup code here, to run once:
9     Wire.begin(2);
10    Wire.onRequest(envia);
11
12    pinMode(bt, INPUT_PULLUP);
13    pinMode(a0, INPUT_PULLUP);
14    pinMode(a1, INPUT_PULLUP);
15 }
16
17 void loop() {
18     // put your main code here, to run repeatedly:
19 }
20
21 void envia(){      //funcao a ser chamada
22     char buf[3];
23     buf[0] = digitalRead(bt);           // le o botao
24     buf[1] = digitalRead(a0);          // le o sensor A0
25     buf[2] = digitalRead(a1);          // le o sensor A1
26     Wire.write (buf, 3);              // Envia o estados dos
                                         // sensores e do botao para o mestre
27 }
```

## 4 COMUNICAÇÃO ETHERNET

Nessa configuração os dispositivos serão divididos em emissor e receptor, o emissor será responsável pelo envio das informações das entradas do sistema, e o receptor sobre o acionamento da válvula. As informações serão enviados sobre um pacote UDP. O esquema físico dessa montagem pode ser visto na [Figura 4.1](#), e a montagem real pode ser observado na [Figura 4.2](#).

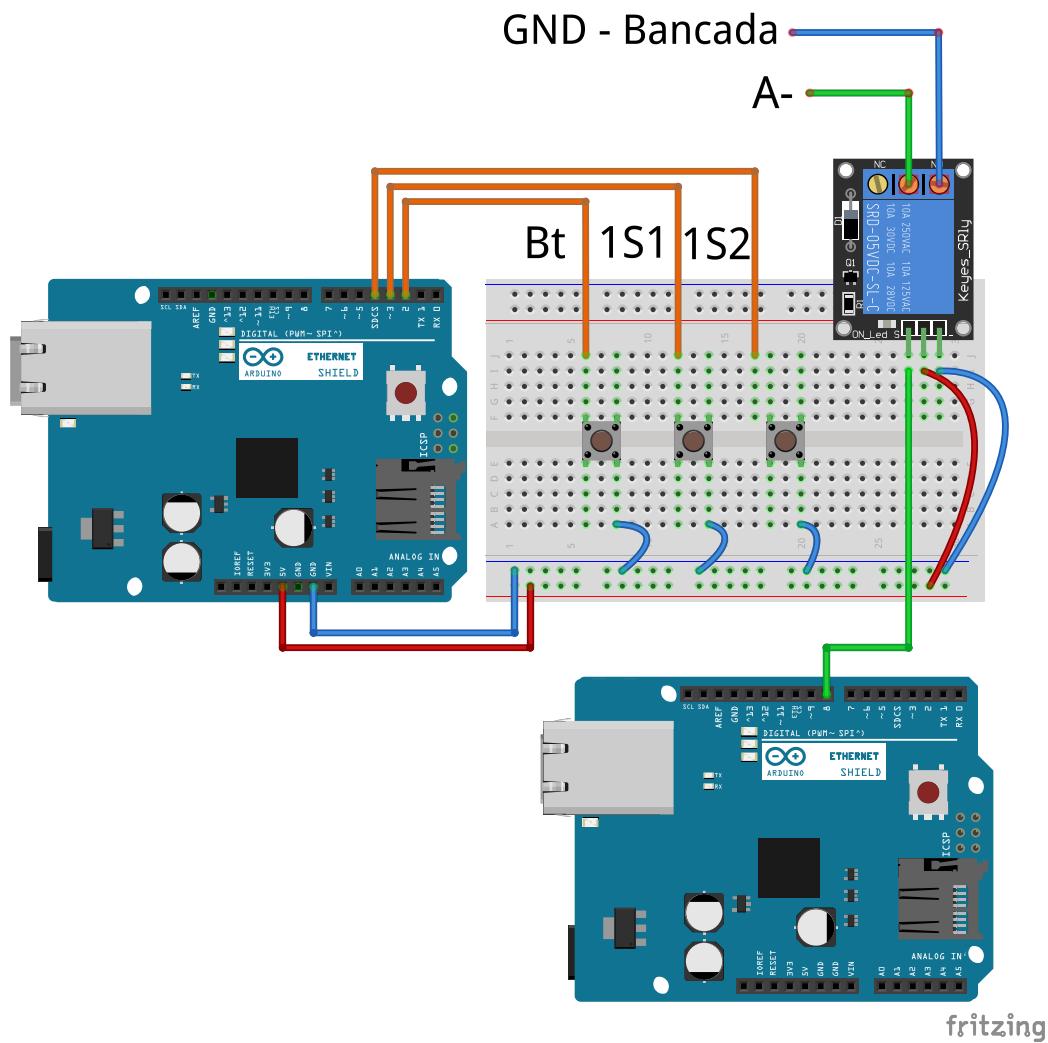


Figura 4.1 – Esquema da montagem da Comunicação Ethernet

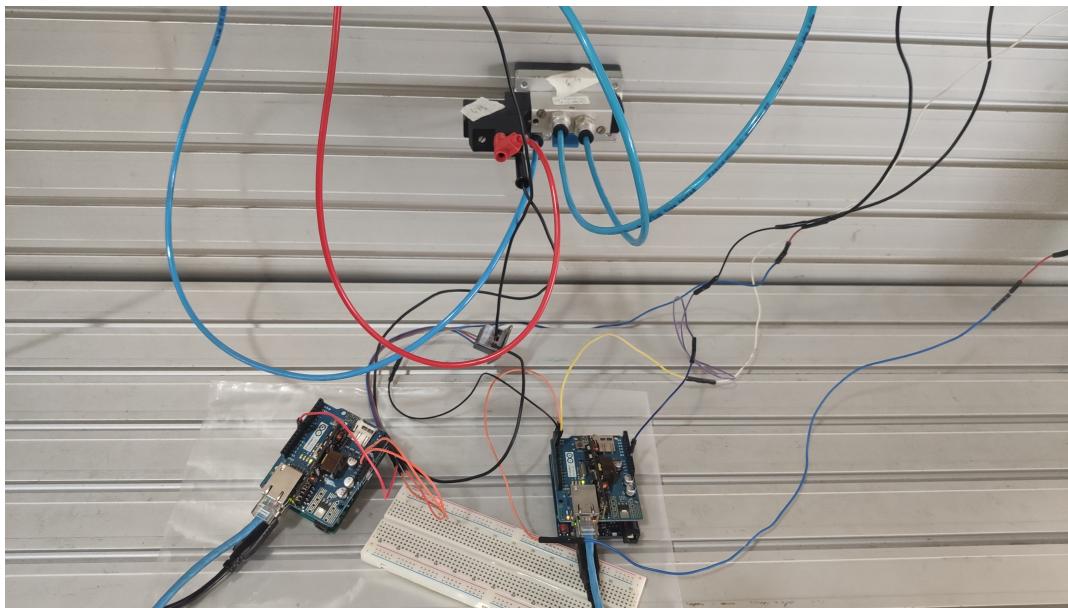


Figura 4.2 – Montagem da Comunicação Ethernet

## 4.1 Emissor

Código 4.1 – Corpo principal Emissor - Ethernet

```
1 #include <SPI.h>
2 #include <Ethernet.h>
3 #include <EthernetUdp.h>
4
5 byte mac[]={0x90,0xA2,0xDA,0x00,0x64,0x44};
6 IPAddress ip(192,168,1,200);
7 unsigned int port =8888;
8 EthernetUDP Udp;
9 byte remoteip[]={192,168,1,80};
10 char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
11
12 const int bt = 2;
13 const int a1 = 3;
14 const int a0 = 4;
15 volatile bool flag = false;
16 volatile unsigned long tempo_disparo = 0;
17 int ultestbt;
18
19 void setup () {
20     Ethernet.begin(mac,ip);
21     Serial.begin(9600);
```

```
22  while (!Serial) {  
23  }  
24  
25  if (Ethernet.hardwareStatus() == EthernetNoHardware) {  
26    Serial.println("Ethernet_shield_n_encontrado.");  
27    while (true) {  
28      delay(1); // n faz nada precisa do shield para iniciar  
29    }  
30  }  
31  if (Ethernet.linkStatus() == LinkOFF) {  
32    Serial.println("Cabo_de_etherent_n_conectado.");  
33  }  
34  Udp.begin(port);  
35  
36  pinMode(bt, INPUT_PULLUP);  
37  pinMode(a0, INPUT_PULLUP);  
38  pinMode(a1, INPUT_PULLUP);  
39 }  
40  
41 void loop(){  
42  // put your main code here, to run repeatedly:  
43  int btt= digitalRead(bt);  
44  int a00= digitalRead(a0);  
45  int a11= digitalRead(a1);  
46  
47  if (btt != ultestbt){  
48    ultestbt=btt;  
49    if(btt==0){  
50      msg_recuo();  
51    }  
52  }  
53  
54  if(a00==0){  
55    if(flag==false){  
56      tempo_disp();  
57    }  
58  }  
59  
60  if (flag){
```

```
61     if(millis()-tempo_disparo >= 4000) {  
62         msg_avanco();  
63     }  
64 }  
65 }
```

Código 4.2 – Envio Mensagem Emissor - Ethernet

```
1 void msg_recuo() {  
2     Udp.beginPacket(remoteip, port);  
3     Udp.write('1');  
4     Udp.endPacket();  
5     tempo_disparo=millis();  
6 }  
7  
8 void msg_avanco() {  
9     Udp.beginPacket(remoteip, port);  
10    Udp.write('0');  
11    Udp.endPacket();  
12    flag = false;  
13 }  
14  
15 void tempo_disp () {  
16     flag=true;  
17     tempo_disparo = millis();  
18 }
```

## 4.2 Receptor

Código 4.3 – Corpo principal Receptor - Ethernet

```
1 #include <SPI.h>
2 #include <Ethernet.h>
3 #include <EthernetUdp.h>
4
5 const int rele = 8;
6
7
8 byte mac[]={0x90,0xA2,0xDA,0x00,0x64,0x50};
9 IPAddress ip(192,168,1,80);
10 unsigned int port = 8888;
11 EthernetUDP Udp;
12 char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
13
14 void setup() {
15   Ethernet.begin(mac,ip);
16   Serial.begin(9600);
17   while (!Serial) {
18   }
19
20   if (Ethernet.hardwareStatus() == EthernetNoHardware) {
21     Serial.println("Ethernet_shield_n_encontrado.");
22     while (true) {
23       delay(1); // n faz nada precisa do shield para iniciar
24     }
25   }
26   if (Ethernet.linkStatus() == LinkOFF) {
27     Serial.println("Cabo_de_etherent_n_conectado.");
28   }
29   Udp.begin(port);
30
31   pinMode(rele, OUTPUT);
32 }
33
34 void loop() {
35   int packetSize = Udp.parsePacket();
36   if (packetSize) {
```

```
37     Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);  
38  
39     if (packetBuffer[0]==‘1’){  
40         recuo();  
41     }  
42     if (packetBuffer[0]==‘0’){  
43         avanco();  
44     }  
45 }  
46 }
```

Código 4.4 – Controle Válvula Receptor - Ethernet

```
1 void recuo(){  
2     digitalWrite(rele, HIGH);  
3 }  
4  
5 void avanco(){  
6     digitalWrite(rele, LOW);  
7 }
```

## Código 4.5 – Acionamento via Serial Receptor - Ethernet

```
1 void serialEvent() {  
2     while (Serial.available()) {  
3         char ls = (char)Serial.read();  
4         Serial.print("Recebido:_");  
5         Serial.print(ls);  
6         Serial.print("\t");  
7  
8         switch (ls) {  
9             case '1':  
10                 recuo();  
11                 Serial.println("Comando_recuo_cilindro");  
12                 break;  
13             case '0':  
14                 avanco();  
15                 Serial.println("Comando_avanco_cilindro");  
16                 break;  
17             default:  
18                 Serial.println("Comando_invalido");  
19                 break;  
20         }  
21     }  
22 }
```

## 5 COMUNICAÇÃO WIRELESS

Na comunicação wireless será usado o modulo XBee, o sistema será dividido em emissor e receptor novamente, e o emissor será responsável pelas entradas do sistema, e o receptor pela saída. O esquema físico da montagem pode ser visto na [Figura 5.1](#)

Como o modulo Xbee utiliza da comunicação serial para realizar a troca de mensagens, no momento de carregar o código para a placa o modulo Xbee deve ser removido.

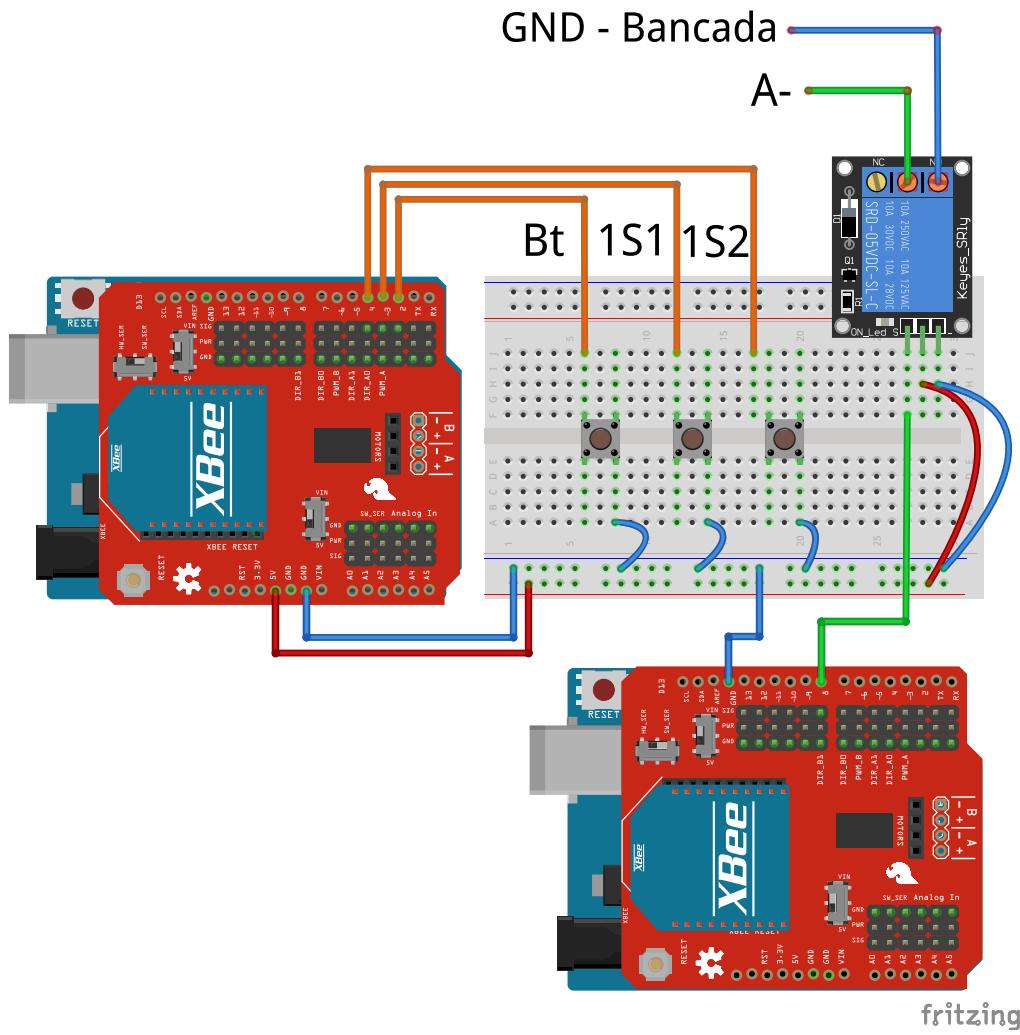


Figura 5.1 – Esquema da montagem da Comunicação Wireless

## 5.1 Emissor

Código 5.1 – Corpo principal Emissor - Wireless

```
1 #include <XBee.h>
2
3 const int bt = 2;
4 const int a1 = 3;
5 const int a0 = 4;
6 volatile bool flag = false;
7 volatile unsigned long tempo_disparo = 0;
8 int ultestbt;
9
10 XBee xbee =XBee();
11 uint8_t payload[]={0};
12 XBeeAddress64 add64 =XBeeAddress64(0x0013A200,0x40C1B208);
13 ZBTxRequest data =ZBTxRequest(add64,payload,sizeof(payload));
14 ZBTxStatusResponse txStatus = ZBTxStatusResponse();
15
16 void setup() {
17   Serial.begin(9600);
18   xbee.setSerial(Serial);
19
20   pinMode(bt, INPUT_PULLUP);
21   pinMode(a1, INPUT_PULLUP);
22
23 }
24
25 void loop(){
26   // put your main code here, to run repeatedly:
27   int btt= digitalRead(bt);
28   int a00= digitalRead(a0);
29   int a11= digitalRead(a1);
30
31   if (btt != ultestbt){
32     ultestbt=btt;
33     if(btt==0) {
34       msg_recuo();
35     }
36   }
```

```
37
38     if(a00==0) {
39         if(flag==false) {
40             tempo_disp();
41         }
42     }
43
44     if (flag) {
45         if(millis()-tempo_disparo >= 4000) {
46             msg_avanco();
47         }
48     }
49 }
```

#### Código 5.2 – Envio Mensagem Emissor - Wireless

```
1 void msg_recuo() {
2     payload[0] = 1;
3     xbee.send(data);
4     tempo_disparo=millis();
5 }
6
7 void msg_avanco() {
8     payload[0] = 0;
9     xbee.send(data);
10    flag = false;
11 }
12
13 void tempo_disp () {
14     flag=true;
15     tempo_disparo = millis();
16 }
```

## 5.2 Receptor

Código 5.3 – Corpo principal Receptor - Wireless

```
1 #include <XBee.h> // inclui a biblioteca
2 XBee xbee =XBee(); // Cria um objeto XBee
3
4 const int rele = 8;
5
6 //Cria um objeto ZBRx na variavel rx
7 ZBRxResponse rx =ZBRxResponse();
8
9 void setup() {
10   Serial.begin(9600);
11   xbee.setSerial(Serial);
12   pinMode (rele, OUTPUT);
13 }
14
15 void loop() {
16
17   uint8_t resposta ;
18   xbee.readPacket();
19
20   //Checando se o pacote foi recebido
21   if(xbee.getResponse().isAvailable()) {
22     if(xbee.getResponse().getApiId()==ZB_RX_RESPONSE) {
23       xbee.getResponse().getZBRxResponse(rx);
24       resposta = rx.getData(0);
25     }
26   }
27
28   if (resposta==1) {
29     digitalWrite(rele, HIGH);
30   }
31   if (resposta==0) {
32     digitalWrite(rele, LOW);
33   }
34 }
```

# REFERÊNCIAS

- 1 BOLLMANN, A. *Fundamentos da automação industrial pneumática: Projetos de comandos binários eletropneumáticos.* [S.l.]: ABHP, 1997.
- 2 TRAINING, P. Tecnologia eletropneumática industrial. *Apostila M1002-2 BR*, 2001.
- 3 ARDUINO. Disponível em: <<https://www.arduino.cc/>>. Acesso em: 25 de dezembro de 2019.